

Velkommen til tjuende gruppetime!

Gruppe 8

Uke 10: Datastrukturer

Tips til objektorientert programmering i IN1000

- *Hvilke klasser og instansvariabler trenger jeg her?*
- *Hvilke metoder trenger jeg i hvilken klasse?*
  - Funksjonalitet og handlinger.
- *Hvordan skal objekter samhandle eller påvirke hverandre?*
  - Avhengigheter.
- Lag gjerne klassediagrammer før dere programmerer, dette kan hjelpe dere med å komme frem til riktig fremgangsmåte å takle oppgavene på

0

(Fortsettelse) Bondesjakk: Eksamensoppgave!

- Nå skal dere starte/fortsette med bondesjakk oppgaven (tre på rad)
- Målet er å løse oppgaven ved å bruke det dere kan om:
  - Klasser og objekter
  - Instansvariabler og metoder
  - Lister og flerdimensjonale strukturer
  - Referanser mellom objekter
  - Testing og feilsøking
- ***Tips:*** *Tegn datastrukturen hvis dere er usikre på hvilke objekter som refererer til hva.*
- Oppgaven finner dere her: https://www.uio.no/studier/emner/matnat/ifi/IN1000/h26/grupper/gruppe-8/Ressurser/10_bondesjakk.pdf

Gjerne spør om hjelp eller jobb i par!

0

Ferdig med bondesjakk-oppgaven?

- Test det med å spille det med en medstudent!
- Du kan også jobbe med:
  - Trix eller innlevering
  - Oppgaver fra tidligere gruppetimer
  - Eksamensoppgaver fra tidligere år
- Hvis alt føles enkelt, prøv et større prosjekt som bruker flere deler av IN1000-pensum.
- Velg gjerne noe du er interessert i, og bygg det på en ryddig måte slik at du kan utvide det senere.

0

Løsningsforslag for Bondesjakk

(ikke sikkert det kjører helt perfekt)

0

Tre-på-rad (Bondesjakk)

I denne oppgaven skal du lage en forenklet utgave av spillet tre-på-rad, også kalt bondesjakk. Du skal gjøre dette ved å benytte objektorienterte prinsipper, altså med klasser og objekter.

Tre-på-rad, eller bondesjakk, foregår på et rutenett av 3*3 ruter. To spillere konkurrerer, og i hvert trekk får de plassere en brikke på rutenettet. Vinneren av spillet er den første spilleren som har tre brikker på rad, enten vertikalt, horisontalt eller diagonalt.

Vi skal bygge programmet vårt fra bunnen og opp (bottom-up), og begynner med å modellere spillere og ruter. Vi går deretter over til å bygge opp brettet som helhet.

Introduksjon

Tre-på-rad (Bondesjakk)

I denne oppgaven skal du lage en forenklet utgave av spillet tre-på-rad, også kalt bondesjakk. Du skal gjøre dette ved å benytte objektorienterte prinsipper, altså med klasser og objekter.

Tre-på-rad, eller bondesjakk, foregår på et rutenett av 3*3 ruter. To spillere konkurrerer, og i hvert trekk får de plassere en brikke på rutenettet. Vinneren av spillet er den første spilleren som har tre brikker på rad, enten vertikalt, horisontalt eller diagonalt.

Vi skal bygge programmet vårt fra bunnen og opp (bottom-up), og begynner med å modellere spillere og ruter. Vi går deretter over til å bygge opp brettet som helhet.

Introduksjon

0

Oppgave A: Klassen Spiller

Lag en klasse *Spiller*. En spiller skal representeres på brettet enten med symbolet X eller symbolet O, dette angis ved opprettelse av en ny spiller. Om symbolet som gis ikke er X eller O skal det printes en beskjed om at symbolet er ugyldig.

Oppgave A

0

Oppgave A: Klassen Spiller

Lag en klasse *Spiller*. En spiller skal representeres på brettet enten med symbolet X eller symbolet O, dette angis ved opprettelse av en ny spiller. Om symbolet som gis ikke er X eller O skal det printes en beskjed om at symbolet er ugyldig.

Oppgave A

0

```python
## oppgave A
class Spiller:
    def __init__(self, symbol):
        if symbol not in ("X", "O"):
            print("ugyldig symbol, oppgi X eller O")
            self._symbol = None
        else:
            self._symbol = symbol

```

Oppgave A

0

Oppgave B: Klassen Rute

Lag en klasse *Rute*. En rute representerer en enkelt rute på et tre-på-rad-brett. Ruter er enten ledige eller opptatt, og om de er opptatt har de en referanse til spilleren som har plassert brikken sin der. Når en rute opprettes settes denne referansen til *None*.

Ruter har en metode *plasser_brikke(self, spiller)* som setter *spiller*-referansen til spilleren som gis som parameter.

Ruter har en metode *er_opptatt(self)* som returnerer *True* dersom ruten er tatt av en spiller, ellers *False*.

Ruter har en metode *hent_brikkeeier(self)*. Denne returnerer spiller-objektet som ligger i spiller-referansen, eventuelt *None*.

Oppgave B

0

Oppgave B: Klassen Rute

Lag en klasse *Rute*. En rute representerer en enkelt rute på et tre-på-rad-brett. Ruter er enten ledige eller opptatt, og om de er opptatt har de en referanse til spilleren som har plassert brikken sin der. Når en rute opprettes settes denne referansen til *None*.

Ruter har en metode *plasser_brikke(self, spiller)* som setter *spiller*-referansen til spilleren som gis som parameter.

Ruter har en metode *er_opptatt(self)* som returnerer *True* dersom ruten er tatt av en spiller, ellers *False*.

Ruter har en metode *hent_brikkeeier(self)*. Denne returnerer spiller-objektet som ligger i spiller-referansen, eventuelt *None*.

Oppgave B

0

```python
## oppgave B
class Rute:
    def __init__(self):
        self._opptatt = False
        self._spiller = None

    def plasser_brikke(self, spiller):
        self._spiller = spiller
        self._opptatt = True

    def er_opptatt(self):
        return self._opptatt

    def hent_brikkeeier(self):
        return self._spiller
```

Oppgave B

0

Oppgave C: Klassen Brett

Lag klassen *Brett*. Et brett er et rutenett på 3*3 ruter, hvor hver rute representeres ved et objekt av klassen *Rute*. Datastrukturen er en nøstet liste, og rute-objekter opprettes i konstruktøren. Konstruktøren oppretter også to referanser til spiller-objekter som i utgangspunktet settes til *None*.

Skriv en metode *legg_til_spiller(self, symbol)*. Metoden oppretter et *Spiller*-objekt med symbolet gitt som parameter, og setter inn dette objektet i en ledig spiller-referanse. Om ingen spiller-referanser er ledig, printes en beskjed om at det allerede eksisterer to spillere.

Skriv en metode *plasser_brikke(self, spiller, x, y)*. Spilleren gitt i parameteren forsøker å plassere en brikke i ruten på plass [x][y]. Sjekk om det er mulig å plassere en brikke på denne plassen, og send i så fall korrekt spiller-referanse til ruten i posisjonen. Om ruten allerede er tatt, printes en beskjed om dette.s

Oppgave C

0

Oppgave C: Klassen Brett

Lag klassen *Brett*. Et brett er et rutenett på 3*3 ruter, hvor hver rute representeres ved et objekt av klassen *Rute*. Datastrukturen er en nøstet liste, og rute-objekter opprettes i konstruktøren. Konstruktøren oppretter også to referanser til spiller-objekter som i utgangspunktet settes til *None*.

Skriv en metode *legg_til_spiller(self, symbol)*. Metoden oppretter et *Spiller*-objekt med symbolet gitt som parameter, og setter inn dette objektet i en ledig spiller-referanse. Om ingen spiller-referanser er ledig, printes en beskjed om at det allerede eksisterer to spillere.

Skriv en metode *plasser_brikke(self, spiller, x, y)*. Spilleren gitt i parameteren forsøker å plassere en brikke i ruten på plass [x][y]. Sjekk om det er mulig å plassere en brikke på denne plassen, og send i så fall korrekt spiller-referanse til ruten i posisjonen. Om ruten allerede er tatt, printes en beskjed om dette.s

Oppgave C

0

```python
## oppgave C
class Brett:
    def __init__(self):
        self._brett = []

        for x in range(3):
            rad = []
            for y in range(3):
                rad.append(Rute())
            self._brett.append(rad)

        self._spillerX = None
        self._spillerO = None

    def legg_til_spiller(self, symbol):
        ny_spiller = Spiller(symbol)

        if symbol == "X" and self._spillerX is None:
            self._spillerX = ny_spiller
        elif symbol == "O" and self._spillerO is None:
            self._spillerO = ny_spiller
        else:
            print("to spillere eksisterer allerede")

    def plasser_brikke(self, spiller, x, y):
        if x < 0 or x >= 3 or y < 0 or y >= 3:
            print("plassen er utenfor brettet")
            return False

        if self._brett[x][y].er_opptatt():
            print("ruten er allerede tatt")
            return False

        self._brett[x][y].plasser_brikke(spiller)
        return True
```

Oppgave C

0

Oppgave D: sjekk_vinner

Skriv en metode *sjekk_vinner(self)* i klassen *Brett*. Denne sjekker alle ruter horisontalt og vertikalt, og returnerer *None* dersom ingen har vunnet. Om en spiller har tre brikker på rad i noen av retningene, returnerer den symbolet til spilleren som har vunnet.

Ekstra: Sjekk også om en spiller har tre brikker på rad diagonalt.

Oppgave D

0

Oppgave D: sjekk_vinner

Skriv en metode *sjekk_vinner(self)* i klassen *Brett*. Denne sjekker alle ruter horisontalt og vertikalt, og returnerer *None* dersom ingen har vunnet. Om en spiller har tre brikker på rad i noen av retningene, returnerer den symbolet til spilleren som har vunnet.

Ekstra: Sjekk også om en spiller har tre brikker på rad diagonalt.

Oppgave D

0



```python
    ## oppgave D
    def sjekk_vinner(self):
        b = self._brett
        linjer = []

        # Rader og kolonner
        for i in range(3):
            linjer.append(b[i])
            linjer.append([b[0][i], b[1][i], b[2][i]])

        # Extra: Diagonaler
        linjer.append([b[0][0], b[1][1], b[2][2]])
        linjer.append([b[0][2], b[1][1], b[2][0]])

        for linje in linjer:
            spiller = linje[0].hent_brikkeeier()

            if spiller is not None:
                if (linje[1].hent_brikkeeier() == spiller and
                    linje[2].hent_brikkeeier() == spiller):
                    return spiller._symbol

        return None
```

Oppgave D

0

Oppgave E: spill

Skriv en metode *spill(self)* i klassen *Brett*. Denne gjør følgende:

Sjekker at det eksisterer to spillere.

I en løkke:

- Lar spillerne vekselvis forsøke å plassere brikke ved *input* av x- og y-koordinat på spillebrettet. Om valget er ugyldig (utenfor brettet eller opptatt rute) skal spilleren få forsøke igjen.
- Hver gang en brikke er plassert, sjekkes det om noen har vunnet. Om det finnes en vinner, printes vinnerens symbol og spillet er over.
- Om alle ruter er utfylt og det ikke finnes en vinner, printes det «Ingen vinner», og spillet avsluttes.

Oppgave E

Oppgave E: spill

Skriv en metode *spill(self)* i klassen *Brett*. Denne gjør følgende:

Sjekker at det eksisterer to spillere.

I en løkke:

- Lar spillerne vekselvis forsøke å plassere brikke ved *input* av x- og y-koordinat på spillebrettet. Om valget er ugyldig (utenfor brettet eller opptatt rute) skal spilleren få forsøke igjen.
- Hver gang en brikke er plassert, sjekkes det om noen har vunnet. Om det finnes en vinner, printes vinnerens symbol og spillet er over.
- Om alle ruter er utfylt og det ikke finnes en vinner, printes det «Ingen vinner», og spillet avsluttes.

Oppgave E

0



```python
    ## oppgave E
    def spill(self):
        if self._spillerX is None or self._spillerO is None:
            print("to spillere må registreres")
            return

        spillere = [self._spillerX, self._spillerO]
        tur = 0

        while True:
            spiller = spillere[tur % 2]
            print("spiller", spiller._symbol)

            x = int(input("rad (0-2): "))
            y = int(input("kolonne (0-2): "))

            if not self.plasser_brikke(spiller, x, y):
                continue

            vinner = self.sjekk_vinner()
            if vinner is not None:
                print("vinner!!:", vinner)
                break

            full = True
            for rad in self._brett:
                for rute in rad:
                    if not rute.er_opptatt():
                        full = False

            if full:
                print("ingen vinner...")
                break

            tur += 1
```

Oppgave E

0